

---

**Algorithm 1** MMMvII( $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ ,  $\alpha$ )

---

```
1: maxMatches =  $\emptyset$ 
2: maxMatchSize = 0
3:  $\delta = (1 + \alpha)/(1 - \alpha)$ 
4: for all  $v_i \in \mathbf{V}$  do
5:   {Build list of  $v_i$ 's neighbours sorted by RPD of their edge to  $v_i$ }
6:    $\mathbf{L} = \{l_a \in \mathbf{V}: (v_i, l_a) \in \mathbf{E}, R(v_i)(l_a) \leq R(v_i)(l_{a+1})\}$ 
7:   for  $a = 1$  to  $|\mathbf{L}|$  do
8:      $v_j = l_a$ 
9:     if  $j < i$  then {Avoids visiting an edge twice}
10:      next a
11:    end if
12:     $e_{min} = (v_i, v_j)$  {Assume an edge of minimum RPD}
13:     $\mathbf{H} = (\mathbf{U}, \mathbf{F}) = (\emptyset, \emptyset)$ 
14:    {Build vertex set  $\mathbf{U}$  of subgraph  $\mathbf{H}$ }
15:    for  $b = a + 1$  to  $|\mathbf{L}|$  do {For all vertices ahead of  $v_j$  in list  $\mathbf{L}$ }
16:       $v_k = l_b$ 
17:       $e_{ik} = (v_i, v_k)$ 
18:       $e_{jk} = (v_j, v_k)$ 
19:      if  $R(e_{ik}) \geq R(e_{min}) \cdot \delta$  then
20:        {If  $e_{ik}$  is not compatible with  $e_{min}$ , then this and all further  $v_k$  are guaranteed to fail checks}
21:        break
22:      else if  $R(e_{jk}) = R(e_{min})$  and  $k < i$  then
23:        next b {Averts duplicate results}
24:      else if  $R(e_{min}) \leq R(e_{jk}) \leq R(e_{min}) \cdot \delta$  then
25:        {Both  $e_{jk}$  and  $e_{ik}$  are forward-compatible with  $e_{min}$  – include  $v_k$  in  $\mathbf{U}$ }
26:         $\mathbf{U} = \mathbf{U} \cup \{v_k\}$ 
27:      end if
28:    end for
29:    {Build edge set  $\mathbf{F}$  of subgraph  $\mathbf{H}$ }
30:    for all  $e_{xy} = (v_x, v_y) \in \mathbf{U} \times \mathbf{U}$  with  $y > x$  do {For all pairs of vertices in  $\mathbf{U}$ }
31:      if  $R(e_{xy}) = R(e_{min})$  and ( $x < i$  or  $y < i$ ) then
32:        next  $e_{xy}$ 
33:      else if  $R(e_{min}) \leq R(e_{xy}) \leq R(e_{min}) \cdot \delta$  then
34:         $\mathbf{F} = \mathbf{F} \cup \{e_{xy}\}$ 
35:      end if
36:    end for
37:     $\mathbf{M} = \text{maxCliques}(\mathbf{H})$  {Maximum cliques of  $\mathbf{H}$  are its largest matches}
38:    for all  $m \in \mathbf{M}$  do
39:       $m = m \cup \{v_i, v_j\}$ 
40:      if  $|m| > \text{maxMatchSize}$  then {Keep only the globally largest matches}
41:         $\text{maxMatchSize} = |m|$ 
42:         $\text{maxMatches} = \{m\}$ 
43:      else if  $|m| = \text{maxMatchSize}$  then
44:         $\text{maxMatches} = \text{maxMatches} \cup \{m\}$ 
45:      end if
46:    end for
47:  end for
48: end for
49: return {maxMatches, maxMatchSize}
```

---

---

**Procedure 1** Ostergard( $G = (V, E)$ )

---

```
1:  $\omega \leftarrow 0$  // Initialize max clique size
2:  $Q \leftarrow \emptyset$  // Initialize set of max cliques
3: for  $i = |V|$  to 1 do // Go through vertices in reverse order
4:    $U \leftarrow \emptyset$  // Build a list of  $v_i$ 's neighbours
5:   for  $j = i + 1$  to  $|V|$  do
6:     if  $(v_i, v_j) \in E$  then
7:        $U \leftarrow U \cup \{v_j\}$ 
8:     end if
9:   end for
10:  OstergardRecursive( $\{v_i\}, U$ ) // Recursively expand the clique
11:   $c[v_i] \leftarrow \omega$  // Max clique size for subproblem starting at  $v_i$ 
12: end for
13: return  $\{Q, \omega\}$ 
```

---

---

**Procedure 2** OstergardRecursive( $q, U$ )

---

```
1: if  $U = \emptyset$  then
2:   if  $|q| > \omega$  then // Found new largest clique?
3:      $\omega \leftarrow |q|$ 
4:      $Q \leftarrow \{q\}$ 
5:   else if  $|q| = \omega$  then // Modification to record all max cliques
6:      $Q \leftarrow Q \cup \{q\}$ 
7:   end if
8: else
9:   while  $U \neq \emptyset$  do
10:    if  $|q| + |U| < \omega$  then // Bound based on remaining vertices
11:      return
12:    end if
13:     $i \leftarrow \min \{j : u_j \in U\}$ 
14:    if  $|q| + c[u_i] < \omega$  then // Bound based on previous subproblems
15:      return
16:    end if
17:     $U \leftarrow U \setminus \{u_i\}$ 
18:     $U' \leftarrow \{u' \in U : (u_i, u') \in E\}$ 
19:    OstergardRecursive( $q \cup \{u_i\}, U'$ )
20:    // Unmodified Ostergard also returns here if Line 2 was true in this
       or any child recursive iterations. Since we must find all max cliques,
       this early exit does not apply.
21:  end while
22: end if
```

---